3) Il problema della ricerca

Nell'informatica esistono alcuni problemi particolarmente rilevanti, poiché essi:

- si incontrano in una grande varietà di situazioni reali;
- appaiono come sottoproblemi da risolvere nell'ambito di problemi più complessi.

Uno di questi problemi è la ricerca di un elemento in un insieme di dati (ad es. numeri, cognomi, ecc.).

Iniziamo a definire un po' più formalmente tale problema definendone l'input e l'output:

- **Input:** un vettore *A* di *n* numeri ed un valore *v*;
- **Output:** un indice *i* tale che *A[i] = v*, oppure un particolare valore ***null*** se il valore *v* non è presente nel vettore.

3.1 Ricerca sequenziale

Un primo semplice algoritmo che viene in mente è ispezionare uno alla volta gli elementi del vettore, confrontarli con *v* e alla fine restituire il risultato. Ci si interrompe appena si trova *v*:

```
Funzione Ricerca (A: vettore; v: intero)
     i ← 1
     while ((i ≤ n) and (A[i] ≠ v))
          i ← i + 1
     if (i ≤ n)
          return i
     else
          return null
```

Oppure:

```
Funzione Ricerca (A: vettore; v: intero)
     i ← 1
     for (i = 1 to n)
          if (A[i] = v) return i
     return null
```

Questo algoritmo ha una complessità di $\Theta(n)$ nel caso peggiore (quando, cioè, *v* non è contenuto nel vettore) e di $\Theta(1)$ nel caso migliore (quando *v* viene incontrato per primo), quindi non abbiamo trovato una stima della complessità che sia valida per tutti i casi. In queste

situazioni diremo che la complessità computazionale dell'algoritmo (in generale, non nel caso peggiore) è un *O(n)*, per evidenziare il fatto che ci sono input in cui questo valore viene raggiunto, ma ci sono anche input in cui la complessità è minore.

Nei casi, come questo, in cui non sia possibile determinare un valore stretto per la complessità computazionale, ed in cui il caso migliore e quello peggiore si discostano, è naturale domandarsi quale sia la complessità dell'algoritmo nel caso medio.

Supponiamo che *v* possa apparire con uguale probabilità in qualunque posizione, ossia che

$$P(v \text{ si trova in i-esima posizione}) = \frac{1}{n}$$

Allora il numero medio di iterazioni del ciclo è dato da:

$$\sum_{i=1}^{n} i\frac{1}{n} = \frac{1}{n}\frac{n(n+1)}{2} = \frac{n+1}{2}$$

Un metodo alternativo è il seguente. Supponiamo che tutte le possibili *n!* permutazioni della sequenza di *n* numeri siano equiprobabili. Di queste, ve ne saranno un certo numero nelle quali *v* appare in prima posizione, un certo numero nelle quali *v* appare in seconda posizione, ecc.

Il numero medio di iterazioni del ciclo sarà di conseguenza:

$$\text{numero medio di iterazioni} = \sum_{i=1}^{n} i\,\frac{\text{numero di permutazioni in cui } v \text{ è in posizione } i}{\text{numero totale di permutazioni}}$$

Ora, il numero di permutazioni nelle quali *v* appare nella *i*-esima posizione è uguale al numero delle permutazioni di (n-1) elementi, dato che fissiamo solo la posizione di uno degli *n* elementi, cioè *(n – 1)!*. Quindi:

$$\text{numero di iterazioni} = \sum_{i=1}^{n} i\,\frac{(n-1)!}{n!} = \frac{1}{n}\sum_{i=1}^{n} i = \frac{n+1}{2}$$

Con un metodo diverso abbiamo trovato lo stesso risultato. Dunque, la complessità del caso medio è un *Θ(n)*.

3.2 Ricerca binaria

E' possibile progettare un algoritmo più efficiente nel caso in cui la sequenza degli elementi sia ordinata (come sono, ad esempio, i cognomi degli abbonati nell'elenco telefonico).

Come cerchiamo un nome nell'elenco telefonico? Iniziamo sempre e comunque dalla prima lettera dell'alfabeto? Ovviamente no… andiamo direttamente a una pagina dove pensiamo di trovare cognomi che iniziano con la lettera giusta; se siamo precisi troviamo il nome nella pagina, altrimenti ci spostiamo in avanti o indietro (di un congruo numero di pagine) a seconda che la lettera del cognome che cerchiamo venga prima o, rispettivamente, dopo quelle delle iniziali dei cognomi contenuti nella pagina.

Un algoritmo che sfrutta questa idea è la ricerca binaria, mostrata nel seguito. Si ispeziona l'elemento centrale della sequenza. Se esso è uguale al valore cercato ci si ferma; se il valore cercato è più piccolo si prosegue nella sola metà inferiore della sequenza, altrimenti nella sola metà superiore.

```
Funzione Ricerca_binaria (A: vettore; v: intero)
    a ← 1
    b ← |A|
    m ← ⌈(a+b)/2⌉
    while (A[m] ≠ v)
        if (A[m] > v)
            b ← m - 1
        else
            a ← m + 1
        if (a > b) return null
        m ← ⌈(a+b)/2⌉
    return m
```

Una prima considerazione: ad ogni iterazione si dimezza il numero degli elementi su cui proseguire l'indagine. Questo ci permette di comprendere dove stia la grande efficienza della ricerca binaria: il numero di iterazioni cresce come *log n*. Il che significa, ad esempio, che per trovare (o sapere che non c'è) un elemento in una sequenza ordinata di un miliardo di valori bastano circa 30 iterazioni!

Per quanto sopra detto abbiamo che la complessità è:

- $\Theta(\log n)$ nel caso peggiore (l'elemento non c'è);
- $\Theta(1)$ nel caso migliore (l'elemento si trova al primo colpo).

Avremo modo in seguito di calcolare formalmente la complessità di $\Theta(\log n)$ del caso peggiore che ora abbiamo individuato solo intuitivamente. Intanto, poiché caso migliore e caso peggiore non hanno la stessa complessità, valutiamo la complessità del caso medio, facendo le seguenti assunzioni:

- il numero di elementi è una potenza di 2 (per semplicità dei calcoli, ma è facile vedere che questa assunzione non modifica in alcun modo il risultato finale);

- v è presente nella sequenza (altrimenti si ricade nel caso peggiore);
- tutte le posizioni di v fra 1 e n sono equiprobabili.

Domandiamoci ora quante siano le posizioni raggiungibili alla *i*-esima iterazione:

- con una iterazione si raggiunge la posizione *i = n/2*;
- con due iterazioni si raggiungono due posizioni: *i =* $\frac{n}{4}$, *i =* $3\frac{n}{4}$;
- con tre iterazioni si raggiungono quattro posizioni: *i =* $\frac{n}{8}$, *i =* $3\frac{n}{8}$, *i =* $5\frac{n}{8}$, *i =* $7\frac{n}{8}$;
- e così via.

In generale, l'algoritmo di ricerca binaria esegue *i* iterazioni se e solo se *v* si trova in una delle 2^{i-1} posizioni raggiungibili con tale numero di iterazioni.

Chiamando *n(i)* il numero delle posizioni raggiungibili con *i* iterazioni, possiamo scrivere che il numero medio di iterazioni è:

$$\textit{numero medio di iterazioni} = \sum_{i=1}^{\log n} \frac{1}{n}\, i * n(i) = \frac{1}{n}\sum_{i=1}^{\log n} i2^{i-1}$$

Ma ricordando che:

$$\sum_{i=1}^{k} i2^{i-1} = (k\text{-}1)2^k + 1$$

otteniamo:

$$\frac{1}{n}\left((\log n - 1)2^{\log n} + 1\right) = \log n - 1 + \frac{1}{n}$$

Ossia, il numero medio di iterazioni si discosta per meno di un confronto dal numero massimo di iterazioni!

Infine, consideriamo una nuova formulazione dell'algoritmo di ricerca binaria (nella quale sono volutamente tralasciati i dettagli per catturarne l'essenza):

```
Ricerca_binaria (A, v)
      se il vettore è vuoto restituisci null
      ispeziona l'elemento A[centrale]
      se esso e' uguale a v restituisci il suo indice
      se v < A[centrale]
            esegui Ricerca_binaria (metà sinistra di A, v)
      se v > A[centrale]
            esegui Ricerca_binaria (metà destra di A, v)
```

L’aspetto cruciale di questa formulazione risiede nel fatto che l’algoritmo risolve il problema ***“riapplicando” se stesso su un sottoproblema*** (una delle due metà del vettore).

Questa tecnica si chiama ***ricorsione***, ed è un argomento importantissimo che sarà illustrato nel prossimo capitolo.